CuEira, gene-enviroment interaction analysis on GPU

Daniel Berglund

April 2014

Abstract

Abstract in english

Abstract

Abstract pÃě svenska

Contents

1 Introduction **1**
- 1.1 Genome-wide association studies . . . 1
- 1.2 Defining Interaction . . . 1
- 1.3 GEIRA and JEIRA . . . 2

2 Mathematical Background **3**
- 2.1 Confounders and Covariates . . . 3
- 2.2 The Multiple Testing Problem . . . 4
- 2.3 Contingency Tables . . . 4
  - 2.3.1 Logisitic Regression . . . 4
  - 2.3.2 Weighted Logistic Regression . . . 5
  - 2.3.3 Statistic Measures . . . 5
- 2.4 Fitting Logisitic Regression Models . . . 5
- 2.5 Data Mining and Machine Learning Approaches . . . 5
  - 2.5.1 Multifactor-Dimensionality Reduction . . . 5
  - 2.5.2 Random Forest . . . 6
- 2.6 Performance Measures . . . 7
  - 2.6.1 Amdahl's Law and Gustafson's Law . . . 7

3 Computer Architecture **10**
  - 3.0.2 CPU . . . 10
  - 3.0.3 Accelerators, GPU and Xeon Phi . . . 10
  - 3.0.4 Which program for GPU and which for CPU? . . . 11
  - 3.0.5 Clusters . . . 12
- 3.1 CUDA programming model . . . 12
  - 3.1.1 Efficient CUDA . . . 12
  - 3.1.2 Profilers . . . 16

4 Algorithm **17**
- 4.1 Current algorithm . . . 17

5 Results **17**

6 Discussion and Conclusions **17**

7 Outlook **17**

8 Appendix **17**

References **18**

1 Introduction

1.1 Genome-wide association studies

Genome-wide association study(GWAS) is a common type of study to search for associations between genetic markers and diseases. Classically it does not consider interactions between the genetic markers nor environmental factors. Investigating gene-gene interactions have started to become more common[1], however gene-environment interactions are still uncommon[2]. Interactions between genes and environmental factors are considered to be important for complex diseases such as cancer and autoimmune diseases.[1, 2, 3, 4]

These type of studies are usually either cohort or case-control. In cohort studies a sample of a population that does not have the disease is followed. Variables that are suspected to be relevant for the disease are measured and over time some of the individuals are getting the disease. The data collected can then be used to find the risk factors. In case-control studies two groups are compared to find the risk factors. One group consists of individuals with the disease and the other of individuals that are similar to the cases but do not have the disease.[5, 6]

The genetic markers are commonly single-nucleotide polymorphisms(SNPs). SNPs are variations in the genome where a single nucleotide is different between individuals in a population[7]. Environmental factors can be various things such as smoking, physical activity and so on. The amount of data to handle usually consists of thousands individuals and hundred thousands or millions of SNPs. Due to the high number of SNPs few algorithms are capable to investigate more than second order interaction in a reasonable time. There are some algorithms that can handle higher interaction orders however these have drawbacks[8, 9, 10].

1.2 Defining Interaction

There are several ways to define interaction. The overall goal is usually to find if *biological* interactions are present. Biological interaction is when the factors cooperate through a physiological or biological mechanism and causes the effect, eg. disease. This information can be used to explain the mechanisms involved in causing the disease and possibly help to find cures for them. However biological interaction is not well defined and thus it is not possible to calculate it directly from data.[5, 11]

Statistic interaction on the other hand is well defined. However it is scale dependent, ie. interactions can appear and disappear based on transformations of the data. Statistic interaction also depends on the used model. The common way to define statistic interaction is to consider the presence of a product term between the factors in the statistical model, this is referred to as *multiplicative* interaction. For instance for a linear model

$$f(x, y) = ax + by + cxy + d$$

c is the product term that represents multiplicative interaction between variables x and y. Statistic interaction is often called just interaction which can

make it a bit confusing.[3, 5]

It can also be defined as the divergence from additive effects on a logarithmic scale, eg

$$OR_{both\ factors\ present} > OR_{first\ factor\ present} + OR_{second\ factor\ present} - 1$$

More about odds ratios(OR) in the section about logistic regression. It's called biological interaction by Rothman[5], sometimes *additive* interaction[3] or *causual interdependence*[12] is used. In this work we will use the term *causal* interaction as in [13].

1.3 GEIRA and JEIRA

GEIRA is a tool for analyzing gene-gene and gene-environment interaction. It uses logistic regression and causal interaction[3]. JEIRA is a parallelized implementation of the environmental interaction analysis in GEIRA. However it can only be run on one node.

2 Mathematical Background

Most research have focused on gene-gene interaction so there are few tools for gene-environment.
SNPs are binary variables which is an advantage that is almost always used when searching for gene-gene interaction. They are binary since the three possible combinations of alleles(AA, Aa, aa) can be represented as 1,1,0 if its dominant model and 0,0,1 if its recessive[].
Environmental factors can be of any type[2]. Gene-gene interaction tools can sometimes be used to find gene-environment interaction, however they usually require the variable to be binary or have other problems since they weren't designed for environmental data.[2]

There are a lot of algorithms and programs proposed for searching for interaction, most have focused on binary variables as already mentioned. They can be roughly classified into four categories, exhaustive, stochastic, machine learning/data mining and stepwise[10]. Both CPU clusters[14] and GPUs[8, 15, 16, 17, 18, 19], in some cases GPU clusters[20] have been used for GWAS. GPUs have been a popular choice since most of the methods for searching for interaction are good on GPUs because each combination can usually be considered independently from the others. More about this in the section 3.0.4.

Exhaustive search is the most direct approach, they compare all combinations of the SNPs in the data. They do not risk missing a combination but it also means they can be slow. Multifactor-Dimensionality Reduction(MDR)[21] and BOOST[22] are two examples.

Stochastic methods uses random sampling to "walk" through the data. BEAM[23] is one example and it uses Markov Chain Monte Carlo(MCMC).

Machine Learning and Data Mining methods are methods from computer science. They are methods learn from data and tries to generalize it. MDR[21] is the among the most common methods used in GWAS that is a computer science approach. See section 2.5 for more details.

Stepwise approaches uses a filtering stage and a search stage. At the filtering stage uninteresting combinations are filtered out by using some exhaustive method. The other SNPs are the examined more carefully in the search stage. BOOST[22] is an example which uses clever data structures and a likelihood ratio test to filter the data.

2.1 Confounders and Covariates

Confounding is one of the central issues in the design of epidemiological studies. It's when the effect of the exposure is mixed with the effect of another variable. So if we do not measure the second variable the first effect would be estimated as stronger than it really is. The second varialbe is then a *confounder*. Several methods in epidemiology are about avoiding or adjusting for confounding. Sometimes these variables needs to be incorporated into the models. Covariates are possible confounders or other variables that you want to adjust for in the

model. Sometimes covariates are called control variables.[11, 5]

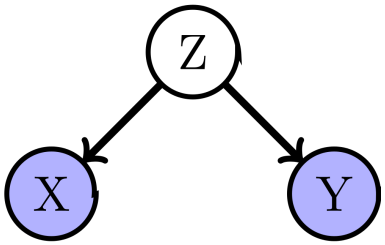

Figure 1: *Illustration of a simple case of confounding. If we do not observe Z we might falesly find an association between X and Y. Wikipedia Commons*

2.2 The Multiple Testing Problem

It's not uncommon to test a lot of hypothesis on the same data and in the case of GWAS it can be billions of tests. If we continue testing we should eventually find something that is significant. Also remember that the standard p-value of 5% means that we expect to find 1 in 20 false positives under the assumption that the null hypothesis is true. The problem arise from the fact that the hypothesis tests are dependent on each other since they use parts of the same data.. This is the multiple testing problem and if it is not corrected for we might find a lot of false positives.[24]

Bonferroni correction is the simplest and viewed as the most conservative way to correct for this problem, simply divided the p-value threshold that a tests passes with the number of hypothesis. However how many hypothesis that are made isn't always obvious. For instance for a two stage analysis. Is the number of hypothesis the number of tests done in both stages combined, the number made in the first stage or the second stage?[24]

2.3 Contingency Tables

A contingency table is a matrix used to describe categorical data. Each cell contains the frequency of occurrences with a specific combination of variables. Table 1 is an example of an 2 x 3 table. From it we can for instance see that 171 persons that got the placebo had an nonfatal attack. Contingency tables are the basis for various statistical tests to model the data. Contingency tables can be used directly for tests like chi^2.[25]

	Fatal Attack	Nonfatal Attack	No Attack
Placebo	18	171	10 845
Aspirin	5	99	10 933

Table 1: *Contingency table describing the outcome of a medical study, from [25]*

2.3.1 Logisitic Regression

For logistic regression the cells of the table are Binomial distributed.[25]

2.3.2 Weighted Logistic Regression

Each individual is assigned a weight that can be any postive number. It's then used to weight the logistic regression.

2.3.3 Statistic Measures

Relative risk and odds ratio[25]
relative risk due to interaction
AP
Causal bonds[13]

2.4 Fitting Logisitic Regression Models

Done with maximum likelihood(ML), often via Newtons method.

2.5 Data Mining and Machine Learning Approaches

Approaches based on Data Mining and Machine Learning have been a popular choice for GWAS. Multifactor-Dimensionality Reduction(MDR)[21] and Random Forest(RF)[26] are among the most common[2, 1]. There are others as well such as clustering approaches [10]. Most of them are used for screening the data for possible interactions[2, 1].

Their biggest advantage is that they are usually nonparametric and designed with high dimensional data in mind. However they are prone to overfitting and the usual way to try to prevent that is to use cross validation and sometimes permutation tests. That means that even if the method itself is fast it is repeated so many times that it can be slow in the end.[1]

2.5.1 Multifactor-Dimensionality Reduction

Multifactor-Dimensionality Reduction(MDR) is a method that reduces the number of dimensions by combining several dimensions into one. For GWAS it combines a number of variables from all the variable combinations and its new dimension is then compared against the outcome and if its predictability is high then the variables that were combined are considered to interact with each other. It goes through all possible combinations of the desired rank like that. It combines the selected n variables by calculating the ratio of cases versus controls for each combination of the possible values of the variables. If the ratio is above a certain threshold all the members of that groups is gets the value 1 for the new dimension, otherwise 0. However due to the cross validation permutation tests it can be slow however it is still faster than exhaustive search with regression methods.[1, 21]

An simple example of MDR using exclusive or(XOR). It's an logical operator that is true if one and only one of it is two variables is true. We have 4 possible combinations and an occurrence for each. The combination (1,0) and (0,1) both have one case with outcome 1 so MDR will classify them as 1 in the new variable Z. The other two combinations have outcome 0 so will be classified with Z=0. And then it is easy to make an predictor from Z to Y.

Y	$\mathbf{X_1}$	$\mathbf{X_2}$
1	1	0
1	0	1
0	0	0
0	1	1

Table 2: *XOR table with outcome* **Y** *and variables* $\mathbf{X_1}$ *and* $\mathbf{X_2}$.

Y	**Z**
1	1
1	1
0	0
0	0

Table 3: *XOR table with* $\mathbf{X_1}$ *and* $\mathbf{X_2}$ *combined into* **Z** *using MDR.*

It can been used for gene-environment interaction but requires modifications since MDR can only handle binary variables. There are extensions that can use continues variables, however they are regression based so they will be slower than regular MDR.[2]

2.5.2 Random Forest

Random Forest(RF) is an ensemble learning method, ensemble methods combine multiple models to improve performance. It takes randomized samples of the data and builds decision trees on each of them. The trees are then combined to form the classifier. Usually hundreds or thousands of trees are used depending on the problem[26]. One of the most popular variants of Random Forest for GWAS is Random Jungle[27].

It has been shown in high dimensional data that it tends to only rank interacting factors high if they have strong marginal effects[28]. Also the ranking of the variables does not say which factor it is interacting with either since it is based on the joint distributions[2]. How to incorporate the environmental factors in RF is also not obvious and using variables with very different scales can bias RFs results[2].
Decision tree figure here

2.6 Performance Measures

An important part in making fast and efficient programs is to know how fast the program is under certain conditions and which parts of the program are slow. For instance the speed could suddenly drop when we start using to many threads, there might be a bottleneck in the network, and so on.

There are two ways to measure how long a program takes to execute. Execution time, sometimes called wall clock time, is how long real life time the program took. The other is to measure the number of processor cycles spent. A parallel program will have shorter execution time than it is serial version however it will likely have spent more processor cycles due to overhead from communication and initialization of the threads. We are usually interested in execution time however number of cycles can be useful for comparison of algorithms.

Speedup is measures how much faster then program is with a certain number of threads compared to the serial version. It's defined as

$$S(p) = \frac{T(1)}{T(p)}$$

Where $T(1)$ is execution time of serial program and $T(p)$ is execution time of parallel program with p threads. Linear speedup is when S(p)=p, it is sometimes called perfect speedup.

Efficieny reflects how efficient the program is using p threads. Linear speed has efficency 1. It's defined as

$$E(p) = \frac{S(p)}{p} = \frac{T(1)}{pT(p)}$$

Strong scaling refers to how the program handles a fixed problem size and increased number of processors. An program with strong scaling has linear speedup. Weak scaling refers to the execution time of the program when there is a fixed problem size *per processor* and the number of processors is increased.[29]

It can be a good idea to plot these measures while varying p, this can show when bottlnecks occur. Looking at the measures at node level can also be useful to get an idea of how increased number of nodes and therefore increased communication affects the performance.

2.6.1 Amdahl's Law and Gustafson's Law

Amdahl's Law is used to find the expected speed up of a system when parts of is parallelized. Simply it says that as the number of processors increases the parts that aren't parallelized will start taking up more and more of the wall clock time and that the speedup for adding more processors will decrease as more and more processors are added and more time is spent relatively on the non parallelized part. It's closely related to strong scaling.[29, 30]

It says that the expected speedup with F fraction of the code parallelized and p threads is

$$S(p) = \frac{1}{(1-F) + \frac{F}{p}(1-F)}$$

As the number of threads grow towards infinity $S(p)$ converges on $\frac{1}{1-F}$. If we have 90% of a code parallelized then even with infinite number of threads we won't get a better speedup than ten.[30]

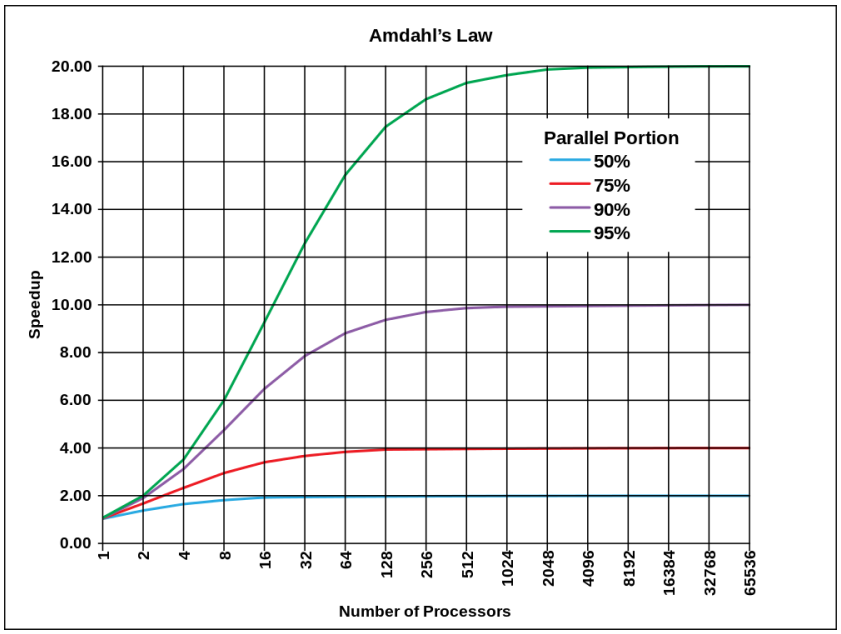

Figure 2: *Illustration of Amdahls Law. Wikipedia Commons*

There are limitation to Amdahl's Law since it makes a couple of assumptions.

- The number of executing threads remain constant over the course of the program.
- The parallel portion has perfect speedup. Often not true due to shared resources,eg caches, memory bandwidth, and shared data.
- The parallel portion has infinite scaling, not true due to similar limits as above. More threads will not increase performance after a while or might even decrease it.
- There is no overhead for creation and destruction of threads.
- The length of the serial portion is independent of the number of threads. Often the serial work is to divided the work to the threads, this work will obviously increase as the number of threads go up. More threads can also lead to more communication overhead.
- The serial portion can't be overlapped by the parallel parts. For instance with producer consumer type pattern the consumer could be strictly serial but the time it takes could be overlapped by the parallel producers.

This means it is most accurate with programs that are of the fork-join type. Eg serial parts and then matrix operations.

Gustafson's Law is closely related to Amdahl's Law and can in some ways be more accurate than Amdahl's Law. Gustafson's Law makes similar assumptions as Amdahl's Law however it also makes two additional statements. It states that problems tends to expand when provided with more computational power, eg increased precision by reducing grid size, high frame rate for graphics. The second is that the parallel portion of the program tends to expand faster than the serial part, eg for matrix multiplication the initialization scales linearly with the matrix size while the multiplication itself scales as $O(n^3)$. The former means that it is closely related to weak scaling. So in a way it says that the execution time remains constant rather than the amount of data. More precise it says that the expected speedup with p threads and F fraction of the code that is parallel is[31, 29]

$$S(p) = p + (1 - F)(1 - p)$$

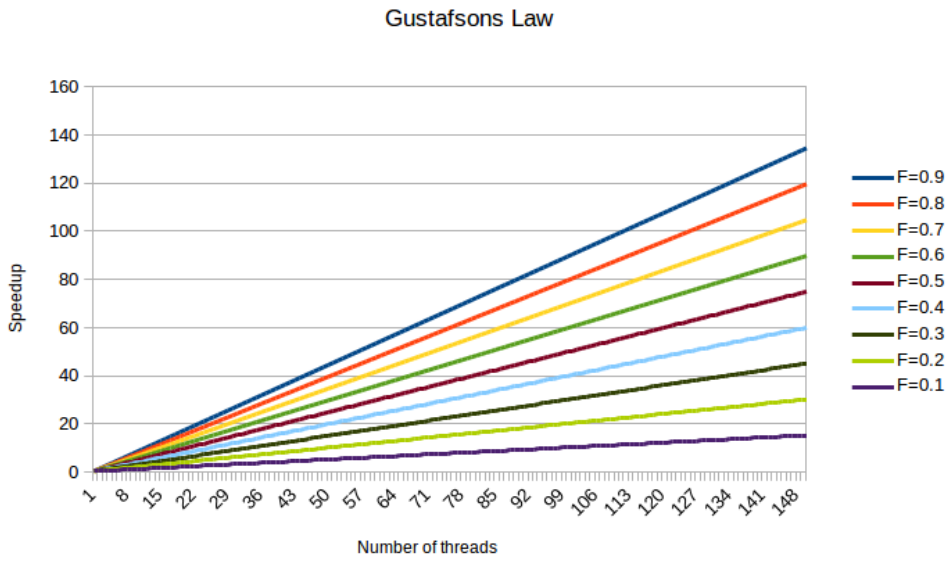

Figure 3: *Illustration of Gustafsons Law*

3 Computer Architecture

Most computers today have a multicore architecture means several processors share the main memory and other resources. they can cooperate on the same problem but it they are coordinated on a case by case basis since each algorithm requires different sharing.
Von Neumann architecture It was first used in EDVAC which was one of the first stored program computers[32]. Stored program computers uses electronic memory to store the program instructions[33].

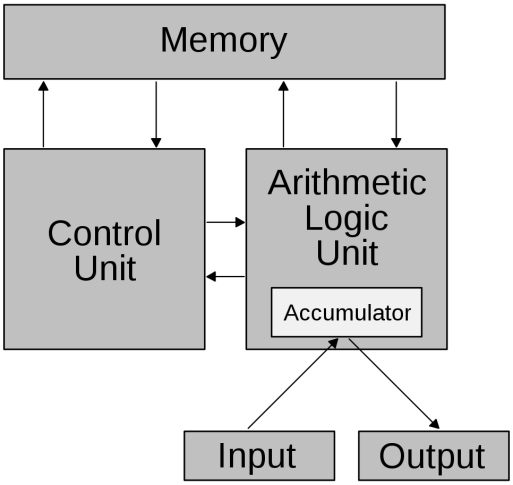

Figure 4: *Schematic of the Von Neumann architecture. Wikipedia Commons*

blabla some text here about von neaumann architecture
For HPC it is easiest to think about the architecture as three main parts, data storage, processor and the memory.[34]

3.0.2 CPU

Central processing unit(CPU) is [35]
Until 1996 there was computers with just a single core CPU on top 500[36]. Since about 2005 most desktop computers CPU has several cores which has increased the need for parallelization. The cores themselves aren't getting faster, the CPUs have more cores.
schematic here
branch prediction prefetching
gpus might be hard but cpus aren't trivial either for HPC parallelization isn't that easy either, depends on problem correct use of caches can be important[37]

3.0.3 Accelerators, GPU and Xeon Phi

Graphics processing unit(GPU) have become more popular for general computing the last 10 years.
something about single and double precision
structure of gpu

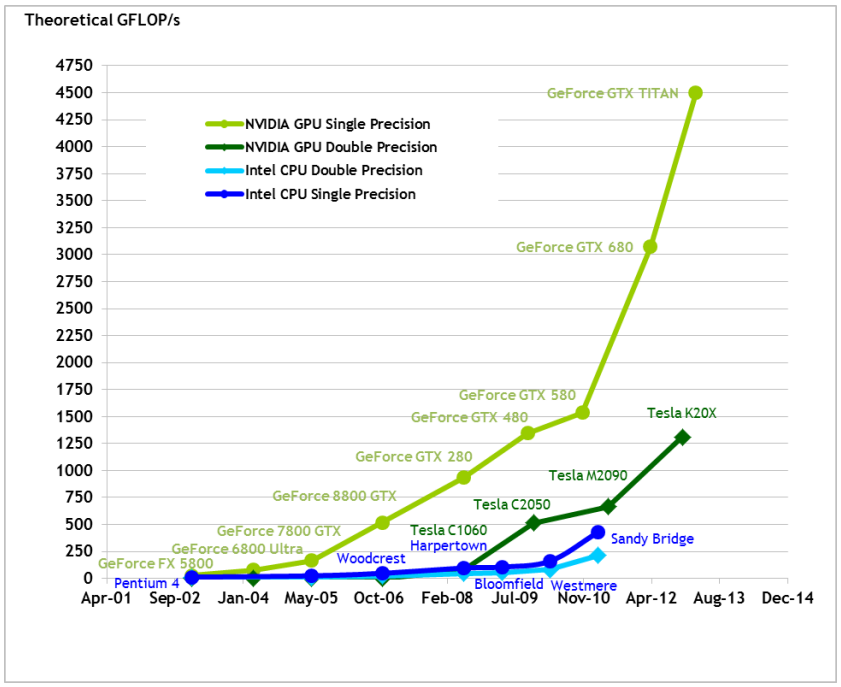

Figure 5: *Bla bla [38]*

Xeon Phi, intel mic mimd knights landing, interesting new development from intel. however usually but not always require memory copy still need vectorisation and other stuff you end up with similar problems might be useful for old serial legacy code, but you still need to parallelize it to get speed
structure of mic

3.0.4 Which program for GPU and which for CPU?

This a big question. As with a lot of things it depends on the algorithm to implement etc.
NVIDIA vs Intel performance claims, intel gets much lower than Nvidia. But who can you trust really?
If you have lots of legacy code then then you might need to rewrite a lot. However by using a profiler the parts that take up the most time can be found and moved to the GPU. However that migh be hard if the program isn't properly structured. No legacy code can also be a good thing. Old code might not be written with modern standards or use new nifty features and therefore lose speed.
Xeon Phi is an alternative, simpler but can still be tricky and require memory copy.
GPUs loses more when going from single precision to double than CPUs. Peak performance on tesla-kepler cards for single precision is around 4 Tflops while double precision is around 1.4 Tflops. [39]
ref the table thing how well does phi do?
In short GPUs excel at linear algebra but Intel MIC is probably a good alter-

native for it to.
GPUs are good for interaction algorithms since most approaches are embarrassingly parallel since they consider each combination independently from the others. They also often perform a lot of linear algebra. Several studies got high gains from implementing their programs on GPU, however their CPU versions weren't parallelized so it is likely that the gains would have been less compared to an optimized and parallelized CPU version.[8, 15, 16, 17, 18, 19] However one study made a CPU cluster version and a GPU version of an algorithm that uses chi^2 tests, they found that 16 CPU nodes had the same performance as a single GTX 280 card[40].

3.0.5 Clusters

Clusters are collections of computers that can work together in several ways and are so tightly connected that they can usually be viewed as a single system. They communciate over a shared network but have seperate memory and procesors. Storage is usually shared. A computer in a cluster is called a *node*. [34, 41]
top 500[36]
Few clusters used GPUs before 2009 but now mix of GPUs and cpu are common among the top clusters. It was driven largely by demand for power saving while still giving high performance. The GPUs can do the heavy computation while other parts are used on CPU. Most of these clusters are highly ranked in Green 500. [41]
Flops is not everything, bad performance compare to theoretical maximum Stuck in network, hard drives, etc. Latency Very few programs scale to the fastest clusters, peta scale.
Tianhe 2 is current top. Theoretical performance 54.9 PetaFlops.[36] Max achived with LinPack 33 PetaFlops 60% efficiency
Picture of Tianhe-2 here
mpi is commonly used for clusters[41]

3.1 CUDA programming model

warps threads and stuff about CUDA[38]
SIMT
blocks
Heterogeneous programming model is used when seperate memory for CPU and GPU.

3.1.1 Efficient CUDA

One of the main criticism against GPUs for general computing purposes is that it is hard to get good performance because it requires good knowledge about details of the GPU architecture, especially the memory architecture. Here are somethings that needs to be considered when writing GPU programs.[19, 38, 29]

Maximize parallelism
: Structure the program and the algorithm in such a way that it is as parallel as possible and overlap the serial parts on CPU with calculations on the GPU.[19, 38]

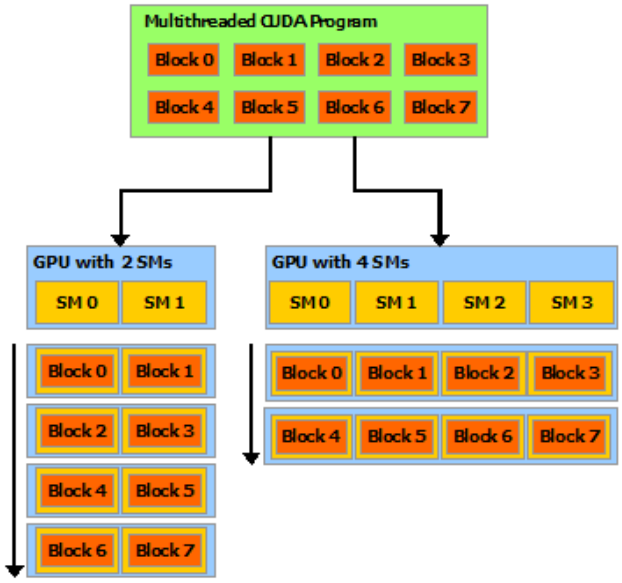

Figure 6: *Text abou blocks. Need to make a black white verison*

Minimize transfers between host and device

Moving data between host and device is expensive and should be avoided if possible. It can be better to run serial parts on the GPU rather than moving the data to the host to do the calculation on the CPU. The bandwidth between host and device is one of the large performance bottlenecks. This can be a problem when the data is to large to fit in the relatively small GPU dram.[38, 29]

Find the optimal number of blocks and threads

There are a lot of things affected by the number of blocks and threads so they should be considered carefully. It's a good idea to parameterize them so that they can be changed for future hardware and varied for optimization. NVIDIA has an occupancy calculator which can be helpful in determining the optimal numbers, however high occupancy does not mean high performance.[38, 29]

The number of blocks should be larger than the number of multiprocessors so that all multiprocessors have at least one block to execute. Having two blocks or more per multiprocessor can be good so that there are blocks that aren't waiting for a ___syncthreads() that can be executed. However this is not always possible due to shared memory usage and similar.[29]

The number of threads per block should be a multiplier of 32 but minimum 64. It's also important to remember that multiple concurrent blocks can reside on the same multiprocessor. To large number of threads in a block and parts of the multiprocessor might be idle since there aren't a block small enough to use those threads. Between 128 and 256 threads is a good place to start.[29]

Streams, concurrent kernels and asynchronous transfers

By using streams it is possible to overlap memory transfers with calculations. This means that the data for the next batch can be transfered while the current batch is calculated and when it is done it can start calculating on the next batch directly after the current one is done. This can hide the time for transfers completely in some situations. However it requires the memory on the host to be pinned. Pinning too much memory can reduce performance of the host.[38]

Operations on a stream is guaranteed to be done in the order they are made however calls on different streams can be executed in any order. Some GPUs allow kernels from different streams to be executed concurrently which can be a big boost if the kernels small enough to fit on the multiprocessors at the same time. Upto 16 or 32 kernels, depending on GPU, can be executed at the same time.

However it requires tweaking of the code that is specific to the GPUs architecture. For instance FERMI architecture only has one queue for the kernels which means that we can get false dependency between kernels. If we have a batch of small kernels and issue them on stream one first and then stream two all the kernels in stream two will be stuck in the queue waiting for the kernels in stream one to finsish, this is because the second kernel on stream one has to wait for the first kernel since they are in the same stream. The second kernel then blocks the queue from moving forward so it will not see the kernels on stream two that could be executed. Kepler GPUs on the other hand has several queues so it won't be a problem on those GPUs. It's the same with asynchronous transfers, some GPUs only have one copy engine while others have one engine for device to host and one for host to device.[38, 42, 43]

Use the correct memory on the GPU

Correct use of caches and memory is important for CPU[37]. It's also important on GPUs but it gets more complicated since the caches are smaller and there are several types of memory.

- Registers
- Global memory is the main memory of the GPU and is accessible from all threads and blocks. However it is relatively slow to access.
- Shared memory is shared inside a block and is faster than global memory. However it is limited in size.
- Constant memory is small however its cached so it is fast to access. It's read only and best used for small amount of variables that all threads access.
- Local memory is tied to the threads scope, but it is still resides off-chip so it has the same access time as global memory.
- Texture memory is read only and can be faster to access than global memory in some situations. This was more important in older GPUs when global memory wasn't cached.[19, 38]

Avoid divergence

Each thread in a warp executes the same instruction at the same time so if some of threads diverge the rest will be ideal until they are at the same instruction again. This means it is important to use control structures such as if statements carefully to prevent threads from idling.[38, 29]

Avoid memory bank conflicts when using shared memory

Shared memory is divided into equally-sized memory modules called banks that can be accessed at the same time for higher bandwidth. Bank conflicts occur when separate threads access the same bank. On some GPUs it is fine if all threads access the same bank. Bank conflicts are split into as many conflict-free requests as needed.[38, 29]

Use existing libraries

Instead of writing everything from scratch it is usually a good idea to use already existing libraries. Especially when performance is important and most task are non trivial on GPUs so using an already optimized library is a good idea. Some of the most popular libraries for CUDA are:

- CUBLAS: BLAS implementation for CUDA[44]
- CULAtools: BLAS and LAPACK implementation for CUDA for both dense and sparse matrices[45]
- MAGMA: BLAS and LAPACK implementation among other things that can distribute the work on both CPU and GPU[46]
- Thrust: Template based library that tries to emulate C++ standard library[47]

Avoid slow instructions

There are some instructions that can be slow and should be avoided if possible, for instance type conversion, integer division and modulo. If a function is called with a floating point number that might be used as a double and require a conversion. By putting an f at the end of the number it is told to be single precision float, for instance 0.5f. In some cases it is possible to use bitwise operations instead which is faster.[38, 29]

Restricted pointers can give increased performance

Aliasing is a problem in some languages, it happens when two pointers point to the same place in memory and the code does operations on them at the same place. This means that operations that look like they can be done in any order actually can't and thus prevents the compiler to make optimizations. The problem is that the compiler has to take this in to account for all pointers even if no aliasing occurs because it might happen. By using the ___restrict___ keyword on pointers the compiler can be told that no aliasing will occur, however it is up to the programmer to make sure that is the case or there might be strange results. Not using aliasing reduces the number of memory accesses the CPU needs to make. However it increases register pressure so it can have an negative effect on performance.[38] Aliasing can be important to keep in mind for CPU code as well[37].

Use fast math functions if precision isn't needed

There are two versions of runtime math functions. The ones of the type funcf() is slower but more accurate than __funcf(). The option -use_fast_math makes the compiler change all the funcf() to __funcf().[29]

3.1.2 Profilers

There are applications called profilers that are made to assess the programs performance and resource consumption. They calculate some of the measures mentioned earlier and they also check hardware usage and how much time the program spends at various parts of the program. This is very useful for finding bottlenecks and other problems in the program. It does not matter if the algorithm is super fast if all the data is stuck in network transfers. The profilers can be hardware dependent so the manufactures usually provided them for their products. For instance NVIDIA provides profilers for programs that use CUDA.[35, 29]

4 Algorithm

4.1 Current algorithm

JEIRA uses Java and Javas built-in functions for concurrency.
Producer consumer pattern
Strucuture picture here.
Symmetric confidence interval of additive effects.

5 Results

6 Discussion and Conclusions

7 Outlook

8 Appendix

References

[1] H. J. Cordell, "Detecting gene–gene interactions that underlie human diseases," *Nature Reviews Genetics*, vol. 10, no. 6, pp. 392–404, 2009.

[2] S. J. Winham and J. M. Biernacka, "Gene–environment interactions in genome-wide association studies: current approaches and new directions," *Journal of Child Psychology and Psychiatry*, vol. 54, no. 10, pp. 1120–1134, 2013.

[3] B. Ding, H. Källberg, L. Klareskog, L. Padyukov, and L. Alfredsson, "Geira: gene-environment and gene-gene interaction research application," *European Journal of Epidemiology*, vol. 26, no. 7, pp. 557–561, 2011.

[4] H. Mahdi, B. A. Fisher, H. Källberg, D. Plant, V. Malmström, J. Rönnelid, P. Charles, B. Ding, L. Alfredsson, L. Padyukov, *et al.*, "Specific interaction between genotype, smoking and autoimmunity to citrullinated α-enolase in the etiology of rheumatoid arthritis," *Nature genetics*, vol. 41, no. 12, pp. 1319–1324, 2009.

[5] K. Rothman and S. Greenland, *Modern Epidemiology*. London: Lippincott Williams and Wilkins, 1998.

[6] C. Mann, "Observational research methods. research design ii: cohort, cross sectional, and case-control studies," *Emergency Medicine Journal*, vol. 20, no. 1, pp. 54–60, 2003.

[7] M. Fareed and M. Afzal, "Single nucleotide polymorphism in genome-wide association of human population: A tool for broad spectrum service," *Egyptian Journal of Medical Human Genetics*, vol. 14, no. 2, pp. 123 – 134, 2013.

[8] B. Goudey, D. Rawlinson, Q. Wang, F. Shi, H. Ferra, R. M. Campbell, L. Stern, M. T. Inouye, C. S. Ong, and A. Kowalczyk, "Gwis-model-free, fast and exhaustive search for epistatic interactions in case-control gwas," *BMC genomics*, vol. 14, no. Suppl 3, p. S10, 2013.

[9] G. Fang, M. Haznadar, W. Wang, H. Yu, M. Steinbach, T. R. Church, W. S. Oetting, B. Van Ness, and V. Kumar, "High-order snp combinations associated with complex diseases: efficient discovery, statistical power and functional interactions," *PloS one*, vol. 7, no. 4, p. e33531, 2012.

[10] S. Leem, H.-h. Jeong, J. Lee, K. Wee, and K.-A. Sohn, "Fast detection of high-order epistatic interactions in genome-wide association studies using information theoretic measure," *Computational Biology and Chemistry*, 2014.

[11] K. J. Rothman, *Epidemiology: an introduction*. Oxford University Press, 2002.

[12] S. Greenland and C. Poole, "Invariants and noninvariants in the concept of interdependent effects.," *Scandinavian journal of work, environment & health*, vol. 14, no. 2, pp. 125–129, 1988.

[13] A. Sjölander, W. Lee, H. Källberg, and Y. Pawitan, "Bounds on causal interactions for binary outcomes," *Biometrics*, vol. X, no. X, 2014.

[14] A. Gyenesei, J. Moody, A. Laiho, C. A. Semple, C. S. Haley, and W.-H. Wei, "Biforce toolbox: powerful high-throughput computational analysis of gene–gene interactions in genome-wide association studies," *Nucleic acids research*, vol. 40, no. W1, pp. W628–W632, 2012.

[15] L. S. Yung, C. Yang, X. Wan, and W. Yu, "Gboost: a gpu-based tool for detecting gene–gene interactions in genome–wide case control studies," *Bioinformatics*, vol. 27, no. 9, pp. 1309–1310, 2011.

[16] Z. Zhu, X. Tong, Z. Zhu, M. Liang, W. Cui, K. Su, M. D. Li, and J. Zhu, "Development of gmdr-gpu for gene-gene interaction analysis and its application to wtccc gwas data for type 2 diabetes," *PloS one*, vol. 8, no. 4, p. e61943, 2013.

[17] S. Lee, M.-S. Kwon, I.-S. Huh, and T. Park, "Cuda-lr: Cuda-accelerated logistic regression analysis tool for gene-gene interaction for genome-wide association study," in *Bioinformatics and Biomedicine Workshops (BIBMW), 2011 IEEE International Conference on*, pp. 691–695, IEEE, 2011.

[18] S. Chikkagoudar, K. Wang, and M. Li, "Genie: a software package for gene-gene interaction analysis in genetic association studies using multiple gpu or cpu cores," *BMC research notes*, vol. 4, no. 1, p. 158, 2011.

[19] J. Poznanovic, "Plinkgpu: A framework for gpu acceleration of whole genome data analysis," Master's thesis, School of Informatics, University of Edinburgh, 2010.

[20] Q. Wang and A. Kowalczyk, "Gwis-gs: Gpu-accelerated screening platform for second order genome wide interaction search." `http://on-demand.gputechconf.com/gtc/2013/presentations/S3169-Genome-Wide-Interaction-Search.pdf`, 2013.

[21] M. D. Ritchie, L. W. Hahn, N. Roodi, L. R. Bailey, W. D. Dupont, F. F. Parl, and J. H. Moore, "Multifactor-dimensionality reduction reveals high-order interactions among estrogen-metabolism genes in sporadic breast cancer," *The American Journal of Human Genetics*, vol. 69, no. 1, pp. 138–147, 2001.

[22] X. Wan, C. Yang, Q. Yang, H. Xue, X. Fan, N. L. Tang, and W. Yu, "Boost: A fast approach to detecting gene-gene interactions in genome-wide case-control studies," *The American Journal of Human Genetics*, vol. 87, no. 3, pp. 325–340, 2010.

[23] Y. Zhang and J. S. Liu, "Bayesian inference of epistatic interactions in case-control studies," *Nature genetics*, vol. 39, no. 9, pp. 1167–1173, 2007.

[24] J. M. Bland and D. G. Altman, "Multiple significance tests: the bonferroni method," *Bmj*, vol. 310, no. 6973, p. 170, 1995.

[25] A. Agresti, *Categorical data analysis*. John Wiley & Sons, Second ed., 2002.

[26] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] D. F. Schwarz, I. R. König, and A. Ziegler, "On safari to random jungle: a fast implementation of random forests for high-dimensional data," *Bioinformatics*, vol. 26, no. 14, pp. 1752–1758, 2010.

[28] S. J. Winham, C. L. Colby, R. R. Freimuth, X. Wang, M. de Andrade, M. Huebner, and J. M. Biernacka, "Snp interaction detection with random forests in high-dimensional genetic data," *BMC bioinformatics*, vol. 13, no. 1, p. 164, 2012.

[29] NVIDIA, *NVIDIA CUDA C Best Practices Guide*, v5.5 ed., July 2013.

[30] X.-H. Sun and Y. Chen, "Reevaluating amdahlâĂŹs law in the multicore era," *Journal of Parallel and Distributed Computing*, vol. 70, no. 2, pp. 183–188, 2010.

[31] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[32] J. Von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[33] W. F. Gilreath and P. A. Laplante, *Computer Architecture: A Minimalist Perspective: Dynamics and Sustainability*. Springer, 2003.

[34] S. Almeida, "An introduction to high performance computing," *International Journal of Modern Physics A*, vol. 28, no. 22n23, p. 1340021, 2013.

[35] G. Hager and G. Wellein, *Introduction to high performance computing for scientists and engineers*. CRC Press, 2010.

[36] "TOP500 Supercomputer Site." `http://www.top500.org`.

[37] U. Drepper, "What every programmer should know about memory," 2007.

[38] NVIDIA, *NVIDIA CUDA C Programming Guide*, v5.5 ed., July 2013.

[39] NVIDIA, "Nvidia tesla-kepler product description." `http://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf`.

[40] R. Jiang, F. Zeng, W. Zhang, X. Wu, and Z. Yu, "Accelerating genome-wide association studies using cuda compatible graphics processing units," pp. 70–76, 2009.

[41] D. B. Kirk and W. H. Wen-mei, *Programming massively parallel processors: a hands-on approach*. Newnes, 2012.

[42] Mark Harris, "How to Overlap Data Transfers in CUDA C/C++." `http://devblogs.nvidia.com/parallelforall/how-overlap-data-transfers-cuda-cc/`.

[43] NVIDIA, "Kepler Tuning Guide." `http://docs.nvidia.com/cuda/kepler-tuning-guide`.

[44] NVIDIA, *CUBLAS Library User Guide*, v5.5 ed., July 2013.

[45] "CULA Tools: GPU Accelerated Linear Algebra," 2010.

[46] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense linear algebra solvers for multicore with gpu accelerators," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1–8, IEEE, 2010.

[47] J. Hoberock and N. Bell, "Thrust: A parallel template library," 2010. Version 1.7.0.